PARALLELIZED CLUSTERING OF UNSTRUCTURED TEXT USING TREE BASED COMPRESSION

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree of

INTEGRATED DUAL DEGREE

in

COMPUTER SCIENCE AND ENGINEERING

(With Specialization in Information Technology)

By

ATUL BAGGA

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2011

CANDIDATE'S DECLARATION

I hereby declare that the work being presented in the dissertation work entitled "**Parallelized Clustering of Unstructured Text Using Tree Based Compression**" towards the partial fulfillment of the requirement for the award of the degree of **Integrated Dual Degree in Computer Science and Engineering (with specialization in Information Technology)** submitted to the **Department of Electronics and Computer Engineering, Indian Institue of Technology Roorkee, India** is an authentic record of my own work carried out during the period from May, 2010 to May, 2011 under the guidance and provision of **Dr. Durga Toshniwal, Assistant Professor, Department of Electronics and Computer Engineering, IIT Roorkee**.

I have not submitted the matter embodied in this dissertation work for the award of any other degree and diploma.

Date: 17th June, 2011

Place: Roorkee

[illegible]

(ATUL BAGGA)

CERTIFICATE

This to certify that the work contained in the dissertation entitled "**Parallelized Clustering of Unstructured Text Using Tree Based Compression**" by Atul Bagga of Integrated Dual Degree in Computer Science and Engineering (with specialization in Information Technology), has not been submitted elsewhere for a degree or diploma to the best of my knowledge.

Date: June,2011

Place: Roorkee

[illegible] 17/6/11

Dr. Durga Toshniwal
Assistant Professor
E&CE Department
IIT Roorkee, India

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my heartfelt gratitude to my guide and mentor **Dr. Durga Toshniwal**, Assistant Professor, Department of Electronics and Computer Engineering, Indian Institue of Technology Roorkee, for her trust in my work, her able guidance, regular source of encouragement and assistance throughout this dissertaion work. I would state that the dissertation work would not have been in the present shape without her inspirational support and I consider myself fortunate to have done my dissertaiton under her.

I also extend my sincere thanks to **Dr. S.N. Sinha**, Professor and Head of the Department of Electronics and Computer Enginnering for providing facilities for the work.

I would like to thank all my friends who supported and encouraged me to finsih this work.

Finally, I would like to say that I am indebted to my parents for everything that they have given to me. I thank them for sacrifices they made so that I could grow up in a learning environment. They have always stood by me in everything I have done, providing constant support, encouragement, and love.

[illegible]

ATUL BAGGA

List of Figures

List of Tables

ABSTRACT

Data mining is the extraction of knowledge from large amounts of data. The data can be of various types like medical data, data from sensors, market basket data, or text. When data mining techniques are applied to text databases the process is termed as text mining. Mining comprises of various tasks such as classification, clustering, rule generation etc. Text clustering is thus the process of dividing text documents into groups such that documents in the same group are similar to one another and different from documents in other groups.

Clustering could be performed by different approaches like partitioning, hierarchical clustering, density based methods or grid based methods. Hierarchical methods have an edge over the others because of the presence of natural multi level hierarchies in case of text. Recent research in the field of text clustering shows the use of various hierarchical methods to cluster text. But these methods generally suffer from high complexity of both time and space. And storing these hierarchy trees is a major problem for large databases like unstructured text.

In the present work, we propose to use a hierarchical clustering algorithm. The algorithm is based on a tree based compression to store the hierarchy of clustered documents such that only the summary of each cluster is stored. The summary of cluster is maintained in such a way that we can retrieve all important information like radius of cluster or centroid of cluster from the summary itself. Once the tree is formed in the memory an iterative approach is implemented to allocate clusters to each document. This reduces the space complexity considerably. For the problem of time complexity we parallelize the algorithm on CUDA architecture. Due to high dimensionality of text parallelization produces significant amount of performance gain over the non parallel version.

Table of Contents

CHAPTER 1
INTRODUCTION

1.1 Introduction

Data mining is a relatively new interdisciplinary field of computer science with a great potential to help organizations focus on the most important information in their vast data warehouses. *Data mining refers to extraction or "mining" knowledge from large amount of data* [1]. Data mining automates tasks which would otherwise be impossible to conduct manually in this advanced age when people have huge capacity to store large amounts of data. It uses the technological advancements in computer science to process data and produce information which can be used by businesses to take decisions more effectively.

Data mining commonly involves four classes of tasks:

1. Association rule learning – Searches for relationships between variables. For example a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.
2. Clustering – is the task of discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data.
3. Classification – is the task of generalizing known structure to apply to new data. For example, an email program might attempt to classify an email as legitimate or spam. Common algorithms include decision tree learning, nearest neighbor, naive Bayesian classification, neural networks and support vector machines.
4. Regression – Attempts to find a function which models the data with the least error.

Data mining is applicable to any kind of data repository, as well as to transient data, such as data streams. Thus the scope of examination of data repositories include relational databases, data warehouses, transactional databases, advanced database systems, flat files or Text files, data streams, and the World Wide Web.

Data mining when performed on text databases is termed as **text mining**, thus text mining is a research area that solves the information overload problem by exploiting recent advances in different fields of technology, allowing the automatic extraction of important elements and facts as they appear in huge volumes of unstructured text. Text mining is the discovery of new, previously unknown information, by automatically extracting information from a usually large amount of different unstructured textual resources. [1]

Most of the data available on internet is unstructured text. Web pages, emails, blogs, technical documents, books in digital format, etc. are a form of unstructured text. These documents contain information which is lost in the vast pool of data being produced in digital form. To retrieve this information we need to perform certain mining tasks.

Typical text mining tasks include text classification, text clustering, concept/entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and entity relation modeling (i.e. learning relations between named entities).

The main focus in this work is on the task of clustering. **Cluster analysis** or **clustering** is the assignment of a set of observations into subsets (called *clusters*) so that observations in the same cluster are similar in some sense, and are different from the observations in other clusters. Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters). The clustering problem has been addressed in many contexts and by researchers in many disciplines; this reflects its broad appeal and usefulness as one of the steps in exploratory data analysis.[2]

The point to note is that clustering only forms a group of similar objects without saying why. For very large datasets clustering helps to form relatively small number of groups of objects in the dataset, properties of which could be further studied to find what makes objects belonging to same cluster similar and different from objects in other clusters.

Cluster analysis has wide variety of real world applications including market research, pattern recognition, data analysis, and image processing. In business, clustering can help marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns. In biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations. Clustering may also help in the identification of areas of similar land use in an earth observation database and in the identification of groups of houses in a city according to house type, value, and geographic location, as well as the identification of groups of automobile insurance policy holders with a high average claim cost. In the field of text it can be used to help in grouping documents for information discovery.

Clustering is also called data segmentation in some applications because clustering partitions large data sets into groups according to their *similarity.* Clustering can also be used for outlier detection, where outliers (values that are "far away" from any cluster) may be more interesting than common cases. Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce.

1.2 Motivation

In 2005 in a speech at an annual conference Eric Schmidt, Google CEO, said that, from the data recorded by the search engine, it seems that, at this specific moment, the Internet is made up of 5 million terabytes. Even Google, which is considered currently the best search engine ever, has succeeded to index only 170 terabytes up until now, Schmidt said.

As we know Internet is a collection of web pages, which are in form of HTML text. 80% of the internet data is unstructured text, and this data is increasing. Movie reviews, internet news, e-books, emails and blogs, source of digital text is just too large to manage. Organizing and using this information effectively requires machine learning algorithms which are not only efficient in terms of time and space complexity but also fit in the need for any particular scenario of text mining.

Search engines, Sentiment detection of online reviews, topic detection for text, book suggestion engines like the one used by Amazon are all examples of wide variety to applications for text mining algorithms.

Clustering (division into groups) is an answer to manage large amount of data effectively. May it be any task like to search, to study more effectively, or to organize documents in a better way clustering seems to answer all the problems. Currently a search engine ranks results in order of popularity, it may be so that u wanted to look for a book which shares its name with a popular movie. It would be good if the engine results are displayed as clusters, then we could always move directly to the category we know or expect our result belongs to. Or the problem may be to organize news articles stored in an archive; here we need to categorize the results for better organization so that the search in future is easy.

The above examples make it clear that source of text is large, and clustering the text is an important problem. Also the solution needs to be effective in terms of memory and time complexity because datasets of text can be huge. Text in general tend to form hierarchies, like a news article talking about a Cricket world cup victory may be kept under the head Global news, precisely in the sports section, and further in the Cricket sub category of sports. Therefore while developing an algorithm to cluster text focus should be on choosing algorithms which can find hierarchical clusters.

1.3 Statement of the Problem

The problem statement for the proposed research work is as follows:

"To perform parallelized, hierarchical clustering of unstructured text documents by using tree based compression."

The problem can be divided into the following sub-problems:

- To preprocess unstructured text documents to convert them into numeric vectors.
- To use tree based indexing, forming a hierarchical summarization of data in the main memory.
- To perform hierarchical clustering using the summary of data stored in the main memory.
- To parallelize the computations on similarity measurement to produce speed up.

1.4 Organization of the Dissertation

This report comprises of six chapters including this chapter that introduces the topic and states the problem. The rest of the dissertation report is organized as follows.

Chapter 2 provides a description of text clustering and different methods applied to cluster unstructured texts. After introducing through the basics of text clustering framework we discuss the various advanced methods applied to cluster text and the research going on in this field.

Chapter 3 provides a detailed description of proposed methods for clustering of text documents, starting with the preprocessor for text, the clustering algorithm itself and the parallelization of it on multi core GPUs with the help of CUDA platform.

Chapter 4 gives the brief description of the implementation of the proposed algorithms of various modules discussed in chapter 3.

Chapter 5 In this chapter we discuss the performance of our implementation on a popular text dataset, its comparison with other used methods and the speedup produced by our parallelized version over the normal code.

Chapter 6 concludes the report here we also discuss what are the future prospects of the work and we suggest improvements over our currently applied method.

CHAPTER 2
BACKGROUND AND LITERATURE REVIEW

With large number of documents in digital form available today, there is an increasing need to automate the process of organization of documents. Clustering, as defined earlier helps to organize data into clusters such that data in one cluster is similar to the data in the same cluster and different from the data in other clusters. Document clustering can thus help to organize documents from a pool containing texts on different category of topics.

2.1 Clustering Algorithms

Clustering Algorithms can be broadly classified into 5 main categories- [1]

1. Partitioning based approach
2. Hierarchical Clustering
3. Density based approach
4. Grid-based approach
5. Model based approach

2.1.1 Partitioning methods

Given a database of n objects or data tuples, a partitioning method constructs k partitions of the data, where each partition represents a cluster and k<= n. That is, it classifies the data into k groups, which together satisfy the following requirements: (1) each group must contain at least one object, and (2) each object must belong to exactly one group. This type of method starts by taking some random K cluster centers and improves the partitioning by moving objects from one group to other. Until stability is achieved or a given threshold criteria is met. Popular algorithms in this category are K-means, K-medoids.

2.1.2 Hierarchical methods

A hierarchical method creates a hierarchical decomposition of the given set of data objects. Based on how the hierarchical decomposition is formed hierarchical methods fall under two sub-categories agglomerative (bottom up approach) or divisive (top down approach). The agglomerative approach starts with each object forming a separate group. It successively merges the objects or groups that are close to one another, until all of the groups are merged into one, or until a termination condition holds. The divisive approach starts with all of the objects in the same cluster. In each successive iteration, a cluster is split up into smaller clusters, until eventually each object is in one cluster, or until a termination condition holds.

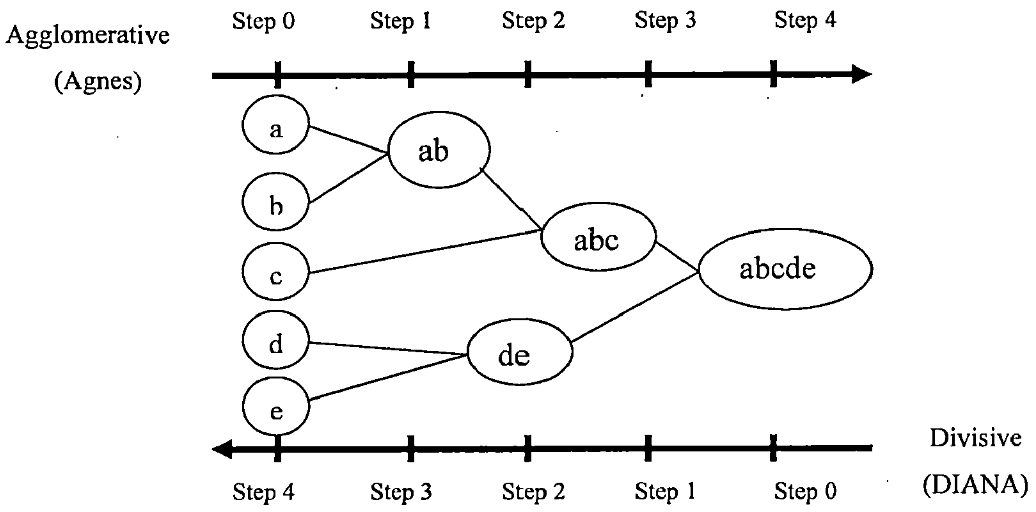


Figure 2.1: Pictorial representation of agglomerative and divisive hierarchical clustering

2.1.3 Density-based methods

Partitioning methods cluster objects based on the distance between objects and can find only spherical-shaped clusters. There are some methods which separate clusters on basis of density. Such methods can find arbitrary shaped clusters too (as shown in figure 2.2). Their general idea is to continue growing the given cluster as long as the density (number of objects or data points) in the "neighborhood" exceeds some threshold. Algorithms like DBSCAN, DENCLUE are two common density based approaches used in this category.

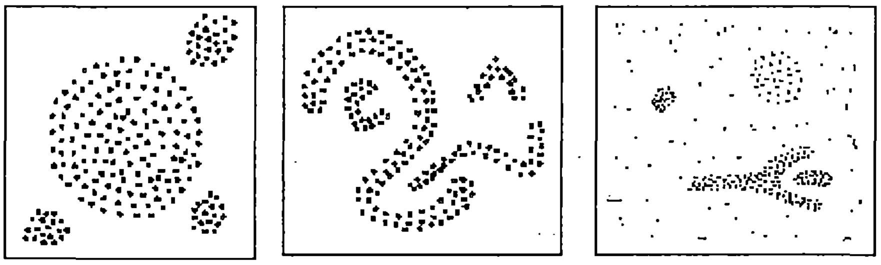


Figure 2.2 Density based clustering can form random shaped clusters

2.1.4 Grid-based methods

Grid-based methods quantize the object space into a finite number of cells forming a grid structure. Clustering is performed on the grid structure or the quantized space. The approach is advantageous because of its fast processing time, which is typically independent of the number of data objects and dependent only on the number of cells in each dimension in the quantized space. STING is a typical example of a grid-based method.

2.1.5 Model-based methods

Model-based methods hypothesize a model for each of the clusters and find the best fit of the data to the given model. A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard statistics. EM is an algorithm that performs expectation-maximization analysis based on statistical modeling. COBWEB is a conceptual learning algorithm that performs probability analysis and takes concepts as a model for clusters.

Not all algorithms can be classified under a particular head, sometimes a hybrid approach works better for a particular type of data. Therefore hybrid algorithms also exist for example an algorithm which may work on a partitioning based approach but also performs hierarchical clustering.

After studying various algorithms used in the field of clustering and going through the needs of a text clustering technique, we discuss here a hierarchical clustering approach based on a tree based compression technique. The algorithm is known as Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

2.2 BIRCH – Balanced Iterative Reducing and Clustering using Hierarchies

First proposed by Zhang, Ramakrishnan and Livny in [3], BIRCH is a relatively new approach to clustering it uses hierarchical clustering at the initial (micro clustering) stage and other clustering methods like iterative partitioning at later (macro clustering) stage. It overcomes two major difficulties of agglomerative clustering methods: scalability and inability to undo what was done in the previous step.

To explain BIRCH in detail we first introduce two concepts- clustering feature and clustering feature tree which are used for representing data clusters in summarized form. These structure help in achieving good speed and scalability in large databases.

2.2.1 Clustering Feature and CF Tree

Given n points in a cluster, where the points are represented in a d-dimensional space. We can define centroid x_0 of cluster as given in eq. 2.1

$$x_0 = \frac{\sum_{i=0}^{n} x_i}{n} \qquad 2.1$$

Radius R of the cluster which can be defined as the average distance of the cluster objects from the centroid of the cluster is given by eq. 2.2

$$R = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}} \qquad 2.2$$

Diameter D of the cluster can be defined as the average pair-wise distance between objects of a cluster and given by eq. 2.3

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n} (x_i - x_j)^2}{n(n-1)}} \qquad 2.3$$

A clustering feature (CF) is a three- dimensional vector summarizing information about clusters of objects. Given *n* *d*-dimensional objects or points in a cluster, then the CF of the cluster is defined as –

$$CF = \langle n, LS, SS \rangle \qquad 2.4$$

Where *n* is the number of points in the cluster, LS is the linear sum of the *n* points (i.e. LS = $\sum_{i=0}^{n} x_i$), SS is the squared sum of the *n* points (i.e. SS= $\sum_{i=0}^{n} x_i^2$),

A clustering feature is a summary of the statistics for the given cluster.

CF vectors follow the additive rule which says:

$$CF_1 + CF_2 = (n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2) \qquad 2.5$$

The CF entry is both compact and accurate at the same time. Compact because it stores much less than all the data points in the sub-cluster and accurate because it is sufficient to calculate all the measurements.

A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering. An example is shown in figure below. By definition, a non-leaf node in a tree has descendants or "children." The non-leaf nodes store sums of the CFs of their children, and thus summarize clustering information about their children. A CF tree has two parameters: *branching factor*, *B*, and *threshold*, *T*. The branching factor specifies the maximum number of children per non-leaf node. The threshold parameter specifies the maximum diameter of sub-clusters stored at the leaf nodes of the tree. These two parameters influence the size of the resulting tree. BIRCH tries to produce the best clusters with the available resources. Given a limited amount of main memory, an important consideration is to minimize the time required for I/O. BIRCH applies a *multiphase* clustering technique: a single scan of the data set yields a basic good clustering, and one or more additional scans can (optionally) be used to further improve the quality [3].

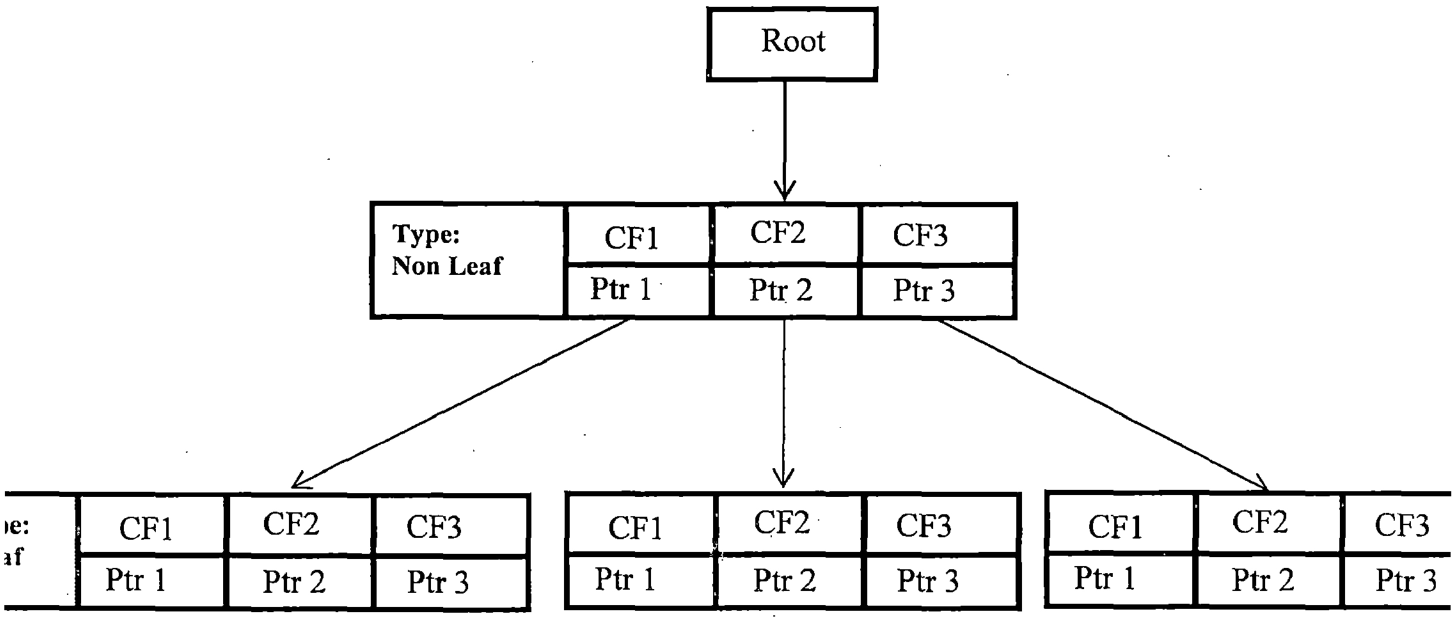


Figure 2.3: Structure of a CF tree, with Height = 1, B = 3

2.2.2 BIRCH Algorithm

Step 1: Identifying the appropriate node- Start from the root and descend the CF-tree, choose the closest child node according to a distance metric like D_0 or D_1..

$$D_0 = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \tag{2.6}$$

$$D_1 = |x_1 - x_0| + |y_1 - y_0| \tag{2.7}$$

where D_0 is centroid Euclidean distance and D_1 is centroid Manhattan distance. $X0_1$ is old centroid $X0_2$ is new centroid.

Step 2: Modifying the leaf- Upon reaching a leaf node, find the closest leaf entry suppose L_i, test if the leaf can absorb the new entry without violating the threshold condition. (That is, the cluster merged with 'Ent' and Li must satisfy the threshold condition.)

If so, update the CF entry for Li to reflect this. If not, add a new entry for 'Ent' to the leaf. If there is space on the leaf for this new entry to fit in, we are done, otherwise we must split the leaf node. Node splitting is done by choosing the farthest pair of entries as seeds, and redistributing the remaining entries based on the closest criteria.

Step 3: Modifying the path to the leaf: After inserting 'Ent' into a leaf, update the CF information for each non-leaf entry on the path to the leaf. In the absence of a split, this simply involves updating existing CF entries to reflect the addition of 'Ent'. A leaf split requires us to insert a new non-leaf entry into the parent node, to describe the newly created leaf. If the parent has space for this entry, at all higher levels, we only need to update the CF entries to reflect the addition of 'Ent'. In general, however, we may have to split the parent as well, and so on up to the root. If the root is split, the tree height increases by one.

2.3 Text Categorization

The following figure shows the basic architecture of any text categorization tool. Our work follows this basic framework. In this section we will discuss the different aspects of the framework in detail.

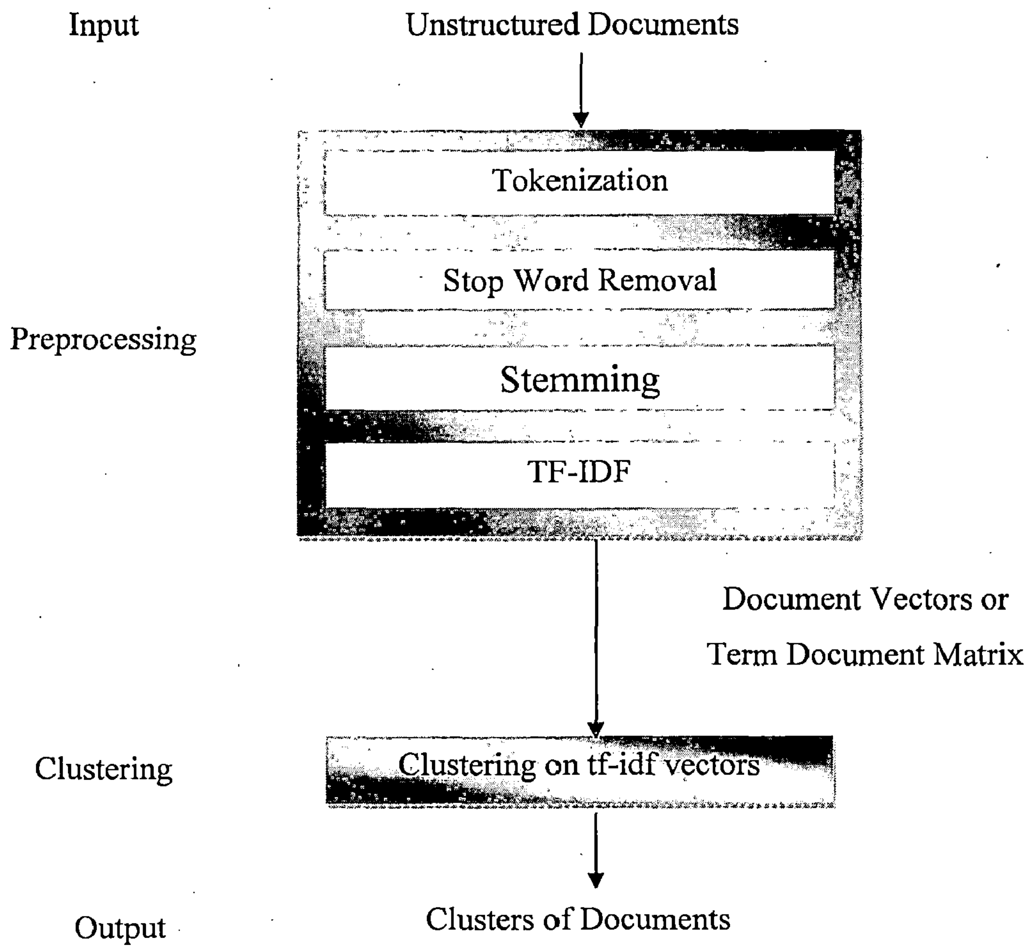


Figure 2.4 Text Clustering framework

To use most clustering algorithms two things are necessary:

- object representation,
- similarity (or distance) measure between objects (unstructured texts in our case)

2.3.1 Representation Model

The above clustering approaches generally work on numeric data, which means the objects to be clustered, are represented as points in an N-dimensional space. For Text categorization we require some similar approach which could convert a simple unstructured document into an object with some numerical representation. Several models have been proposed for text representation, here is a brief description of the most commonly used model.

Vector Space Model

Vector space model (or term vector model) is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as index terms [4]. The basic idea is that we represent the documents as vectors in a high dimensional space corresponding to keywords in the document corpus and use an appropriate similarity measure to compute the similarity between document vectors or between document and query vector [5]. Thus in a set of d documents and containing t terms, we can represent each document as a vector v in a t dimensional space R^t.

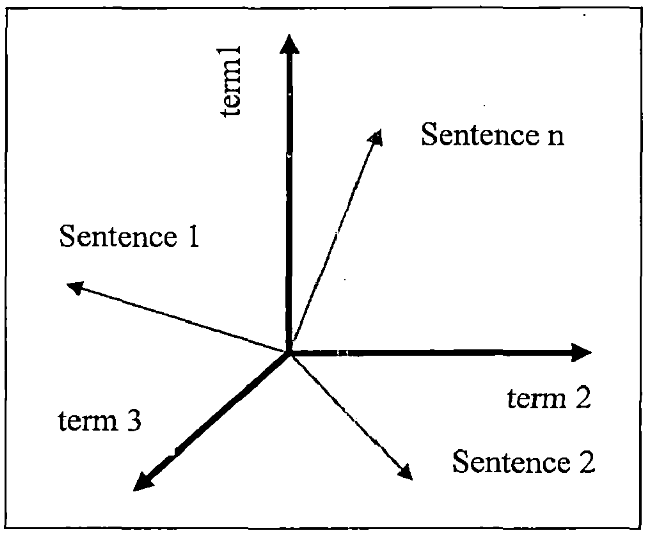


Figure 2.5 Illustration of Vector Space model

Document represented as vector-

$$D_j = (w_1, w_2, \ldots, w_t)$$

Let the term frequency be the number of occurrences of term t in the document d, that is, freq(d,t). The (weighted) term-frequency matrix TF(d, t) measures the association of a term t with respect to the given document d. It is generally defined as 0 if the document does not contain the term, and nonzero otherwise. There are many ways to define the term-weighting for the nonzero entries in such a vector. For example, we can simply set TF(d, t) = 1 if the term t occurs in the document d, or use the term frequency freq(d, t), or the relative term frequency, that is, the term frequency versus the total number of occurrences of all the terms in the document. There are also other ways to normalize the term frequency. For example, the Cornell SMART system uses the following formula to compute the (normalized) term frequency:

$$TF(d,t) = \begin{cases} 0 & if\ freq(d,t) = 0 \\ 1 + log\left(1 + \left(1 + log(d,t)\right)\right) & otherwise \end{cases} \qquad 2.8$$

Besides the term frequency measure, there is another important measure, called inverse document frequency (IDF), that represents the scaling factor, or the importance, of a term t. If a term t occurs in many documents, its importance will be scaled down due to its reduced discriminative power. For example, the term Sports may likely be less important if it occurs in many documents in a sports magazine. According to the same Cornell SMART system, IDF(t) is defined by the following formula:

$$IDF(t) = log\frac{1+|d|}{|d_t|} \qquad 2.9$$

Where d is the document collection, and d_t is the set of documents containing term t. If $|d_t| << |d|$, the term t will have a large IDF scaling factor and vice versa.

In a complete vector-space model, TF and IDF are combined together, which forms the TF-IDF measure:

$$TF\text{-}IDF(d,t) = TF(d,t) * IDF(t) \qquad 2.10$$

2.3.2 Preprocessing Tasks

Before text documents are subjected to a tf-idf weighing measure several other tasks need to be performed for better working of the representation model.

Tokenization: The first and foremost step in the text mining process is to get tokens in a document. That is to break the document into smallest units useful for the algorithm or task to be performed. Generally tokens are relevant words because irrelevant words are removed in the step described below.

Removal of Stop Words: Some common words such as prepositions or conjunctions etc appear in each document many no. of times. These terms have no discriminative power and increase the dimensionality of the Vector Space which in turn results in poorer accuracy and higher complexity of algorithm. Such terms are called Stop Words and they are removed from the document during the text preprocessing phase.

Stemming: A single word can exist in various forms in document corpus but with the same meaning, that is various terms may share a common word stem. A text retrieval system needs to identify the group of words where each word is a small syntactic variant of the root. Like drug, drugged, drugs has the same root drug and can be viewed as variants of the root word drug, So all three forms should be changed to the root word. The process is called stemming.

2.3.3 Similarity Measurement

Thus combined with preprocessing and the representation and weighing scheme we get a model which converts text into a numerical model representation of relevant words. To implement any clustering algorithm we need to have a similarity measure. There are several measures proposed a few of them are as described below:

Minkowski Distance: Minkowski distance is a generalization of distance on Euclidean space. Both Euclidean distance and Manhattan distance as explained next are special cases of minkowski distance which is given by formula shown in eq. 2.11

$$d(i,j) = (|x_{i1} - x_{j1}|^p + |x_{i2} - x_{j2}|^p + \dots\dots\dots + |x_{in} - x_{jn}|^p)^{\frac{1}{p}} \qquad 2.11$$

where p is a positive integer, i and j represent two points and x_is and x_js are coordinates of the points i and j. This distance is sometimes referred to as L_p norm.

Euclidean Distance: The Euclidean distance or Euclidean metric is the distance between two points that one would measure with a ruler, and is given by the Pythagorean formula. By using this formula as distance, Euclidean space (or even any inner product space) becomes a metric space. The associated norm is called the Euclidean norm.
Euclidian distance basically returns the length of the line segment joining the two points in the space.
Euclidean distance is defined by the formula:

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots\dots\dots + (x_{in} - x_{jn})^2} \qquad 2.12$$

Where i and j are two points in the space and x_is and x_js are coordinates of the points i and j.
From the above eq. we can easily notice this is a special case of Minkowski distance, Euclidean distance is Minkowski with p = 2, that is L_2 norm.

Manhattan Distance: Another well-known metric is Manhattan (or city block) distance, defined as:

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots\dots\dots + |x_{in} - x_{jn}| \qquad 2.13$$

Where i and j are two points between which the distance is being calculated and x_is and x_js are coordinates of the points i and j.
Manhattan distance is also special case of Minkowski distance with p = 1 or L_1 norm.

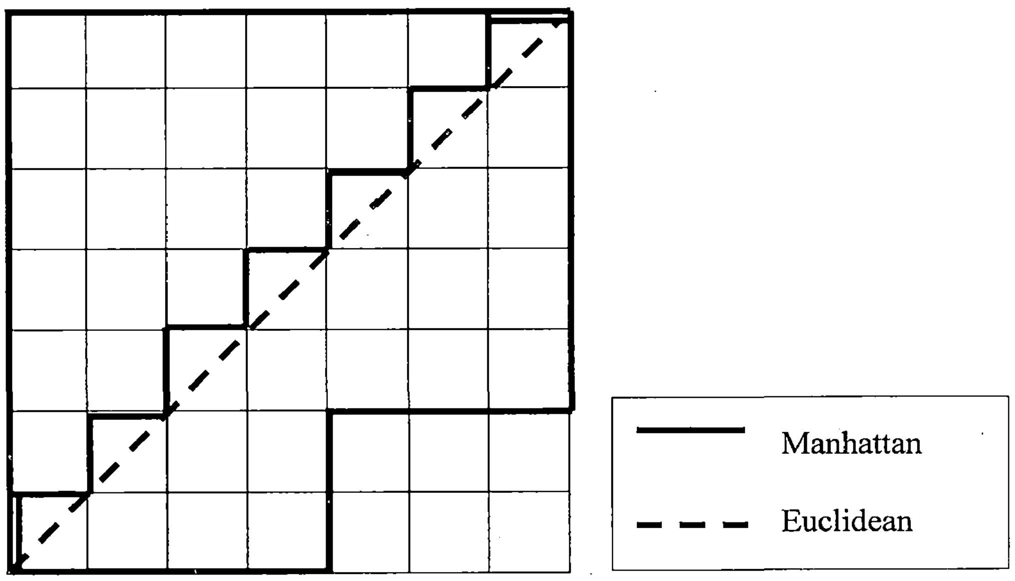


Figure 2.6 Euclidean distance and Manhattan distance.

Cosine Similarity: Cosine similarity [1][5] is a measure of similarity between two vectors by measuring the cosine of the angle between them. The result of the Cosine function is equal to 1 when the angle is 0, and it is less than 1 when the angle is of any other value. Calculating the cosine of the angle between two vectors thus determines whether two vectors are pointing in roughly the same direction.

Cosine of two vectors can be easily derived by using the Euclidean Dot Product formula:

$$a.b = |a||b| \cos\theta \qquad 2.14$$

Given two verctors of attributes, A and B, the cosine similarity, θ, is represented using a dot product and magnitude as:

$$similarity = \cos\theta = \frac{A.B}{|A||B|} = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} * \sqrt{\sum_{i=1}^{n}(B_i)^2}} \qquad 2.15$$

For text matching, the attribute vectors A and B are usually the term frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison.

The resulting similarity ranges from −1 meaning exactly opposite, to 1 meaning exactly the same, with 0 usually indicating independence, and in-between values indicating intermediate similarity or dissimilarity.

In the case of information retrieval, the cosine similarity of two documents will range from 0 to 1, since the term frequencies (tf-idf weights) cannot be negative. The angle between two term frequency vectors cannot be greater than 90°.

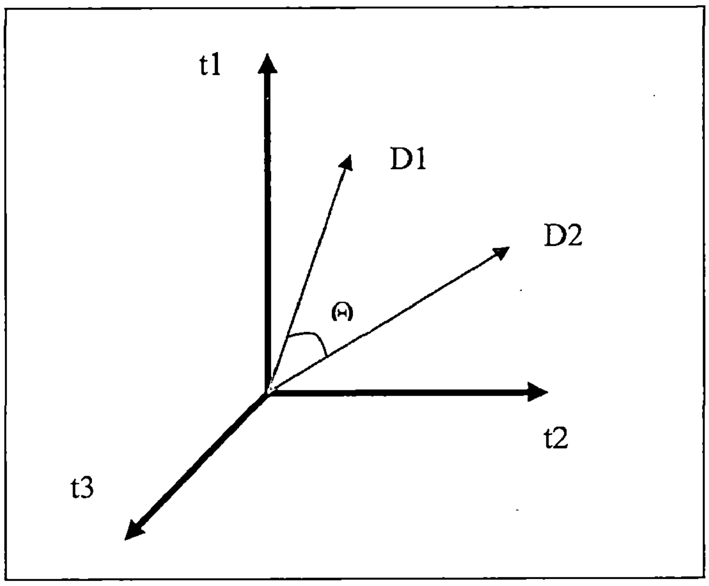


Figure 2.7: Cosine Similarity in Vector Space Model Representation

2.3.4 Quality of Clustering

Accuracy: The most often used measure to denote quality of clustering is the accuracy. Accuracy of clustering is easy to calculate and understand. It is simply the percentage of total documents that were correctly placed in the majority class or dominant class.

$$Accuracy = \frac{\sum_{r=1}^{k} n_{rr}}{n} * 100\% \qquad 2.16$$

Where $n_{rr}$is the number of objects belonging to class C_r placed in clusterS_r.

For Example: Suppose there are two types of flowers that exist and we input data on only one kind of flower but two clusters are formed and 90 out of 100 data points are placed in one cluster, rest in another, then the majority cluster is the rightly spotted class and 90% of the data points are correctly placed into the cluster. The accuracy is 90%.

Given a particular category L_r of size n_r and a particular cluster S_i of size n_i, then n_{ri} is the number of documents that belong to category L_r assigned to cluster S_i.

Precision: Defined as $P(L_r, S_i) = \frac{n_{ri}}{n_i}$, is the ratio of number of documents of category r assigned to cluster i to the total number of documents in cluster i.

Recall: Defined as$R(L_r, S_i) = \frac{n_{ri}}{n_r}$, is the ratio of number of documents of category r assigned to cluster i to the total number of documents in category r.

FScore: F-Score measures the degree of each cluster contain documents from the original category. In F-Score function, R is the recall value and P is the precision value defined for category Lr and cluster Si. It is given by eq. 2.17

$$FScore = \sum_{r=1}^{k} \frac{n_r}{n} . max_{i \le i \le k} \frac{2.R(L_r,S_i).P(L_r,S_i)}{R(L_r,S_i) + P(L_r,S_i)} \qquad 2.17$$

Before we go into the research performed on text clustering algorithms let us have a brief introduction to CUDA.

2.4 Compute Unified Device Architecture (CUDA)

CUDA or Compute Unified Device Architecture [6] is a parallel computing architecture developed by Nvidia. CUDA is the computing engine in Nvidia graphics processing units (GPUs) that is accessible to software developers through variants of industry standard

programming languages. Programmers use 'C for CUDA' with Nvidia extensions and certain restrictions.
CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest Nvidia GPUs become accessible for computation like CPUs. Unlike CPUs however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly. This approach of solving general purpose problems on GPUs is known as GPGPU. CUDA provides both a low level API and a higher level API. CUDA works with all Nvidia GPUs from the G8X series onwards.

CUDA also provides libraries of pre-implemented functions in Java and C++ which are implemented through parallel processing on CUDA GPUs. Through the use of these libraries the development process becomes easy as we only have to use these prebuilt functions in our code to implement a particular task or module in parallelized way. Here is a brief description of such a library for C++. It is called thrust.

Thrust is a C++ template library for CUDA based on the Standard Template Library (STL). Thrust allows us to implement high performance parallel applications with minimal programming effort through a high-level interface that is fully interoperable with CUDA C. Thrust provides a rich collection of data parallel primitives such as scan, sort, and reduce, which can be composed together to implement complex algorithms with concise, readable source code. By describing the computation in terms of these high-level abstractions we can provide Thrust with the freedom to select the most efficient implementation automatically. As a result, Thrust can be utilized in rapid prototyping of CUDA applications

2.5 Literature Survey

Now that we have fair idea of how text clustering works, here is a brief overview of all previous studies and research done on the subject.

Early work on text clustering started in 1971 when Jardine and Van Rijsbergen stated that the cluster hypothesis is fundamental to the issue of improved effectiveness. It states that relevant documents tend to be more similar to each other than to non-relevant documents and therefore tend to appear in same clusters. Thus if the cluster hypothesis holds relevant documents will be separated from the non-relevant ones.

Croft in 1978 proposed the idea of improved searching by using clustering stating that a relevant document may be ranked low in a best match search because it may lack some query terms. In a clustered collection this relevant document may be clustered together with other relevant documents that do have the required query terms and could therefore retrieved in a clustered search.

Initially document clustering was investigated for improvement in precision and recall of an IR system (Rijsbergen et al., 1981) and as an efficient way of finding the nearest neighbors of a document (Buckley and Lewit, 1985). Later it found use in browsing document collection (Cutting et al., 1992) and in organizing results returned by search engine (Zamir and Etzioni, 1998).

Papers published in the field of document clustering generally cover a number of diverse areas like development of efficient clustering algorithms for document clustering (Larsen and Aone, 1999), the application of document clustering to browse large document collections(Hearst and Pederson, 1996), the visualization of clustered document spaces (Allan et al., 2001). Document clustering was also used for clustering documents hierarchically (Koller and Sahami, 1997). Another document clustering approach (Aggarwal et al., 1999) produces an effective document classifier by finding natural clusters in already existing document taxonomy.[7]

Many techniques have been applied to clustering documents; Cutting et al. (1992) [8] provided details on using partition based clustering algorithms to clustering documents. Two of his techniques are Buckshot and Fractionation. Buckshot selects a small sample of documents to pre cluster them using standard algorithms then assigns the other documents to the clusters formed. Fractionation splits N documents into m buckets where each bucket contains N/m documents, each bucket is then clustered using standard partitioning algorithm and n/x clusters are formed where x is input parameter. This process is repeated by treating clusters formed in the previous step as single objects.

There were similar works until a new scheme was proposed by Zamir and Etzioni (1998) [9]. This was when phrase based clustering was introduced for the first time. They used generalized suffix-tree to obtain information about the phrases and used them to cluster the documents.
Most of the work around the year 2000 was focused on the efficiency aspects for online tasks rather than the effectiveness of method.

Methods inspired from the famous K-means clustering such as K-modes (Huang, 1997) [10] also inherited the shortcomings of K-means like dependence on seed clusters and inability to automatically detect the number of clusters.

In [11], Cui and Potok (2005) proposed a PSO based hybrid document clustering algorithm. The algorithm performs a search globally in the entire solution space. They applied PSO, K-means and the hybrid PSO + K-means clustering algorithm on the four different text document datasets in their experiments. The results illustrated that more compact clustering results were obtained using hybrid PSO+K-means than by any method alone.

Csorba and Vajk (2006) [12] presented a novel topic based document clustering technique where there is no need to assign all the documents to the clusters. Under this system the results are much cleaner considering the fact that many documents have ambiguous topic. The method applies a confidence measurement for every classification result and the documents which fall short of the pre-defined minimum confidence limit are removed from the results. Thus the system only clusters documents it feels sure about, rest of them are marked as unsure. By varying the minimum confidence limit the results of document clustering can be filtered strongly in this method.

Jing et al. (2007) [13] presented a novel K-means type method, an algorithm for clustering high dimensional data objects like text in sub-spaces. The basic idea is that clusters are often formed in sub-spaces rather than the entire space. For example, in text clustering clusters of documents of different topics are categorized by different subsets of terms or keywords. Thus the keyword of one cluster may not occur in the other cluster. The new algorithm calculates weight for each dimension in each cluster and uses the weight values to identify the subsets of important dimensions that categorize different clusters.

Sun et al. (2008) [14] developed a novel hierarchical algorithm for document clustering. They used the cluster overlapping phenomenon to design cluster merging criteria. The system computes the overlap rate in order to improve the time efficiency and the veracity and the line passed through the centre of the two clusters instead of the ridge curve. Also based on the hierarchical clustering method it used the EM (expectation maximization) algorithm in the Gaussian mixture model to count the parameters and make the two sub-clusters combined when their overlap is the largest.

Muflikhah and Baharudin (2009) [15] proposed a method that integrates the information retrieval method and document clustering as concept space approach. It used the latent semantic index (LSI) approach which used Singular Value Decomposition (SVD) or Principle Component Analysis (PSA). LSI is used for dimensionality reduction of matrix by finding pattern in document collection. Each method is implemented to weight of term-document in vector model for document clustering.

In [16], Gil-García et al. discussed two approaches dynamic hierarchical compact and dynamic hierarchical star. The first produces disjoint hierarchies while the second produces overlapped hierarchies. Both approaches were useful for a producing a topic hierarchy for document dataset. But the methods suffered from high time complexity.

2.6 Research Gaps

Based on the study presented in the previous section the following research gaps were identified-

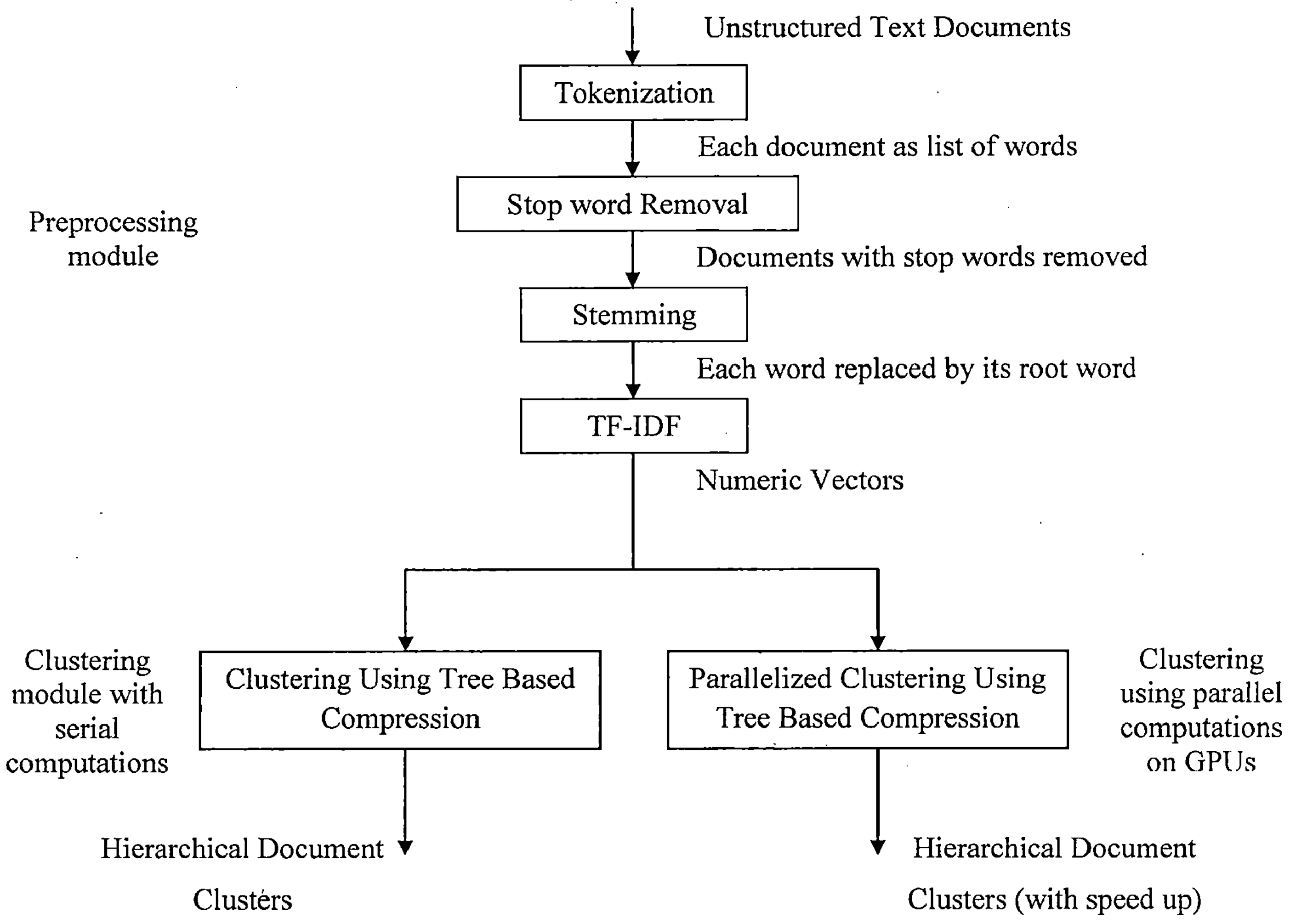
- Clustering algorithms generally assume all data can be accommodated in the memory at the same time, but this is not the case with large datasets. Thus the general clustering algorithms become very inefficient in terms of memory usage. The algorithm used for clustering of text should be efficient in terms of memory usage; use of summary data structures can solve this task.
- Text documents naturally form hierarchical clusters, thus the need is to have a hierarchical clustering algorithm which can form clusters at various levels of hierarchy thereby giving complete information as to which cluster does the document belong to at each level of hierarchy.
- In general the hierarchical clustering algorithms proposed so far have a limitation that time complexity of these algorithms is very high. Also they require large amount of memory to store the hierarchy tree after each step. Therefore the requirement is to have an algorithm which uses some form of summaries to store data in a compact space.

CHAPTER 3

PROPOSED WORK

The problem of Text Clustering has become more important in recent years because of the exponential growth of Internet, and thereby the increase in text resource in digital form and the increasing sophistication of the data mining algorithm to leverage this information.

The basic design of the proposed work is as shown below:



Unstructured Text Documents
Tokenization
Each document as list of words
Stop word Removal
Preprocessing module
Documents with stop words removed
Stemming
Each word replaced by its root word
TF-IDF
Numeric Vectors
Clustering module with serial computations
Clustering Using Tree Based Compression
Parallelized Clustering Using Tree Based Compression
Clustering using parallel computations on GPUs
Hierarchical Document Clusters
Hierarchical Document Clusters (with speed up)

Figure 3.1 Proposed Framework

3.1 Proposed Scheme for Text Categorization

In this chapter, we discuss the methods for Text categorization. The proposed work as shown in the basic framework is divided into 3 modules. Module 1 is the preprocessor for Text, it takes as input the unstructured text format and returns the document in vector form using the Vector space model combined with tf-idf weighing scheme. In Module 2 we implement the clustering algorithm we use for categorization of text, which is a basically a hierarchical clustering algorithm. Module 3 is the parallelization part where we parallelize our proposed algorithm on parallel computing GPUs using CUDA programming model.

3.1.1 Module 1: Text Preprocessor

This module deals with the first and foremost step of any text mining task that is to clean the data and prepare it for use in the algorithm. Preprocessing being a common task, several codes for it are available. But instead of using a prebuilt preprocessor we propose a simple yet effective idea to build a preprocessor for texts.

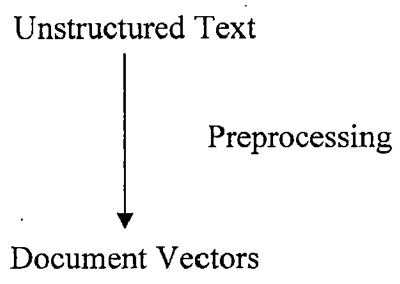


Figure 3.2 Preprocessing

Starting with the basic step of tokenization the document can be read line by line, broken into tokens using punctuations and space as break points. Once the document is converted into a bag of words we check each word for its existence in the stop word list used by the famous SMART information retrieval system. If the word exists in the stop word list we remove it from the corpus vocabulary. Once the documents have been tokenized and stop words have been removed, we stem the words to their root form.

The task of stemming or converting the word to root form can be performed to varying levels. Like a system could consider synonyms belonging to a single word root or sometime just take into account the syntactic root obtained via removal of suffixes like –ing, -s, -es, -ed, -tion etc. The degree of stemming depends on the complexity of the system or performance required from the stemming task. For example- A complex Artificial intelligence system like sentiment analysis could use a complex stemmer which considers synonyms belonging to same word root, while a simple query processing system might just consider a simple suffix removal algorithm sufficient as a stemmer.

For our work we propose to use the Porter's Stemmer [17], originally written by M.F. Porter as a part of larger IR project and later published as "A Program for Suffix Stripping". Code for Porter's stemmer in various languages is available for free use from its website.

The Preprocessor stores the documents in form of a list of stemmed words as temporary files. These files are read by the tf-idf sub module code which stores in a dictionary form the words and their count for each document. At the same time any words which occur in a document very few times are removed from the corpus vocabulary and their count for the document is deleted.

When all the documents have been read we get a vocabulary list which represents the number of distinct words in the corpus, the number of distinct words represents the dimensionality of the vector space we will use to represent each document.

With the word frequency in store we use the SMART IRs tf-idf definitions as given in previous chapter in eq. 2.8, 2.9 and 2.10.

Special points for Preprocessor used:

The weighing scheme is coded such that the words in the introductory paragraph of the text get more weight as compared to words occurring later in the text. This is done because generally the starting lines of any text contain the crisp of the matter talk about in the text. For example a news paper article will start by saying which place, person, or field it is linked to.

The words which occur rarely like a word occurring only 2-3 times in a text does not represent the document's domain. Such words can be removed to reduce the dimensionality of the vector space.

3.1.2 Module 2: Clustering Using Tree based compression algorithm

After studying various algorithms used in the field of clustering and going through the needs of a text clustering technique we suggest here use of a hierarchical clustering approach based on a tree based compression technique. The algorithm is known as BIRCH- Balanced Iterative Reducing and Clustering Using Hierarchies. Details of the algorithm are discussed in chapter 2.

BIRCH is a highly scalable and speedy algorithm [3], so it is well suited for a problem like text clustering where dimensionality is high and datasets are large.

BIRCH is normally well suited for numerical attributes where the similarity metrics of distance works well. In case of text we first need the preprocessor to convert unstructured documents into numerical vectors representing these documents. The normal preprocessing works when documents have to be represented as vectors. In case of vectors distance based similarity does not hold good and we need cosine similarity.

Cosine similarity in case of BIRCH would mean that the summaries won't work to calculate all the measurements of the cluster. Thus we use Euclidean distance as similarity measurement along with a preprocessor designed to weigh terms such that important terms get more weightage. The way we have achieved this is explained in the previous section.

Complexity of BIRCH

BIRCH algorithm makes a single scan of the dataset to insert each vector into the clustering feature tree. Thus the time complexity of the BIRCH algorithm can be given as O(N), for given a total of N documents to cluster. Most of the processing time is spent in making similarity computations so as to decide to which block or group the given vector should be added. Because of the large size of vectors serial implementation effectively has a high time complexity. Suppose the dimensionality of space is n then the complexity of BIRCH is O(N*n).

3.1.3 Module 3: Parallelization on CUDA (platform for parallel computing)

The details about Compute Unified Device Architecture or CUDA are explained in section 2.4. In our work we propose to parallelize the functions on BIRCH algorithm which work on high dimensional vectors. Finding distances between two high dimensional points, adding or subtracting two high dimensional vectors and finding the closest cluster to a given point. All these tasks require small calculations but in large number of data objects which are stored in a vector form. CUDA provides an ideal resource to achieve speed in these tasks by performing these calculations on devices with high parallel computing powers i.e. the Nvidia's GPUs.

We propose here to implement the four functions which perform calculations on these high dimensional vectors to be implemented for parallel calculations through CUDA's support. This implementation could be performed in one of the various ways that is by C programming where we have to write the parallel implementation code by ourselves or by using a language like Java or C++ which has direct support from the CUDA developers, providing libraries like Thrust (explained in section 2.4) which help in easy and fast development of code.

Thus in our module 3 we simply use C++ thrust libraries to implement parallel versions of the four functions listed below.

1. Find Nearest CF: This function finds the nearest (sub-cluster) CF vector to the given vector passed as argument.
2. Calculate Combined CF: This function calculates the combined CF by adding the point passed as argument to the CF and returns the new combined CF value.
3. Add CF: This function adds the two CFs that are passed into the function and returns the combined CF value.
4. Subtract CF: This function subtracts one CF from another passed into the function and returns the difference CF value.

The four functions proposed to be parallelized are called very often and required for each of the entry made in the CF tree. Parallelization of these functions therefore is expected to produce significant improvement in speed of the original algorithm.

CHAPTER 4
IMPLEMENTATION DETAILS

4.1 Implementation

The implementation of the proposed scheme is done on C++ programming language. The Graphical user interface for the first module that is pre-processing of text is designed on .NET. C++ is chosen as the language because of its seamless integration with CUDA, CUDA provides support for C++ in form of THRUST libraries to ease the work of programmers for parallelization of code on CUDA. .NET on the other hand provides support for quicker development of GUIs and automated testing and performance measurement of the same.

4.2 Design and Development of the Text Pre-processor

The module is implemented using C++.NET for the drag and drop GUI design support provided by the Visual Studio IDE for .NET.

Design of the pre-processor is divided into 4 basic sub-modules. The sub-modules are-

1. Tokenization
2. Stop word removal
3. Stemming
4. Tf-idf weighing

<u>Tokenisation</u> is implemented with the basic C++ string token function, the document is read and input into the string token function which returns the array of tokens

<u>Stop Word Removal</u>: For stop word removal we have used the Stop Word list used in the famous SMART IR system. Each token is matched against the list of stop words stored in alphabetical

order, on finding the match the word is removed from the document and a temporary text file is maintained which does not contain any stop words.

Stemming: We use the Porter's stemmer or the famous Porter's Suffix stripping program [15] as a stemmer. The code is freely available under the BSD licence from the website. The program often does not return the correct root but a modified word, but the purpose in our case is only to match all forms of the word to a common word no matter what the common word is taken to be. Being fast due to its low complexity Porter's stemmer perfectly fills the needs of Text Clustering.

Implementation of mathematical model to represent text: we have chosen to use tf-idf weighing scheme with minor adjustments to use as our mathematical model. A regular tf-idf [3] scheme simply calculates the term frequency for each term in the document (number of time a particular term appears in the document) and its inverse document frequency (ratio of total no. of documents to number of documents containing that term).

We have made the following changes:

- Extra weight to terms in the introductory paragraph of text.
    - We have coded tf-idf module in such a way that the terms in the first paragraph get some extra weightage in the frequency count because generally they are the representative terms in the document
- Rarely occurring terms are removed from the documents
    - Terms that occur only 2-3 times in a document do not provide any distinct advantage in similarity measurement of the document. Such terms increase the dimensionality of the space many folds so removing such terms not only improves the accuracy of our algorithm but also makes it fast.
- Only a fixed amount of terms are taken from each document
    - Not all the frequently occurring terms are taken into the vocabulary from a document. Because of our initial rule of providing extra weight to introductory terms we generally have the representative terms quiet distinctly ahead in frequency count. But still in case the document is large and contains many

frequent term we choose among them the first x% of the terms, where x is the threshold vocabulary limit for each document.

4.3 Development of clustering algorithm

The clustering algorithm used here is explained in the previous chapter. Here we will discuss the implementation details of the algorithm.

The code for the algorithm is written in standard C++ programming language. C++ is used because of being simple and still providing the OOPS programming paradigm for making the code easy to understand and reuse. Also CUDA platform provides libraries in C++ to provide parallelization support. Thus our simple C++ code can be parallelized with much ease using the THRUST libraries written for CUDA. This part is explained in the next section of this chapter.

The Class Diagram of the Birch Tree is as shown below:

Member Functions	Description
Int Nearest_Cf(vector<double>, vector<CF>)	Finds the nearest node to Cf vector passed from the vector of CFs passed, and returns the index of nearest node.
bool CheckThreshold(CF)	To check if the nearest node is more than threshold distance from the CF.
bool CheckBranchingFactor(Node*)	To check if the spilt causes the number of nodes at a branch exceed the branching factor.

void UpdateNodesOnPath(stack<Pathnode>, vector<double>)	The function is called when the spilt of nodes occurs. It modifies the upper levels of the tree by adding the node to previous level and managing if more splits occur.
Node* AddNode(Node*, stack<Pathnode>)	Called when a spilt occurs in adding new node. To add the newly formed node to the trees upper level.
CF CalculateCombinedCF (Node*)	Adds the CF to the Node Summary (now the new point is a part of the new Node returned)
CF SubstractCf(CF, CF)	Returns a CF vector which is difference of the two CF vectors passed as arguments
CF AddCf(CF, CF)	Returns a CF vector which is sum of the two CF vectors passed as arguments
vector<Node*> LevelTraverse(vector<Node*>,int)	Function to level traverse the tree, finds cluster centres at each level of hierarchy
void AllocateClusters(vector<vector<double>>,int)	The function to calculate cluster by iterating through the dataset once and write the clusters onto a text file.

Table 4.1: Class Diagram: BIRCH CLUSTERING

4.4 Parallelization of clustering module on CUDA platform

As described in the previous chapter CUDA provides developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Through high and low level APIs it makes parallelization an easy to achieve task.

We have used C++ to code our clustering algorithm so the shortest and quickest way to achieve parallelization and reusing the code was through C++ libraries provided for CUDA support. These are called Thrust Libraries and they provide simple functions like sort, add, and invert etc which have their parallel implementation on the device vector. We simply copy our vector into the device vector and implement our calculations in terms of these functions.

Device vector is a STL container vector which is maintained on the device memory when a function like Thrust::sort() is called on such vector it automatically picks up code from the Thrust libraries at compile time and thus the function is implemented through parallel implementation on CUDA GPUs.

The following functions were parallelized-

1. Int Nearest_Cf(vector<double>, vector<CF>)
   This function finds the nearest (sub-cluster) CF vector to the given vector passed as argument 1. We have parallelized this function, and now calculating distances in parallel calculations on various dimensions.

2. CF CalculateCombinedCF (Node*)
   This function calculates the combined CF by adding the vector point to the CF, here the calculations on adding LS and SS vectors have been implemented through CUDA functions to speed up the run time.

3. CF AddCf(CF, CF)
   This function adds the two CFs that are passed into the function, which requires adding the LS and SS vectors to be added as well this is now implemented in parallel functions provided by the thrust library for CUDA.

4. CF SubstractCf(CF, CF)

   This function subtracts one CF from another passed into the function, this requires subtractions to be performed on very large LS and SS vectors to be subtracted, this is now implemented in parallel functions provided by the thrust library for CUDA.

The four functions parallelized are called very often and required for each of the entry made in the CF tree. Parallelization of these functions therefore produces significant improvement in speed of the original algorithm.

CHAPTER 5

RESULTS AND DISCUSSION

The following sections discuss the performances of proposed algorithms across various parameters. The results for proposed work have been tested over test dataset.

5.1 Dataset Used

There are generally two datasets that are widely used for text mining tasks like classification, clustering and topic detection. They are Reuters 21578 and 20-Newsgroups dataset. They are both a series of news articles taken from different categories.

We have chosen 20-Newsgroup dataset, the primary reason being the presence of hierarchical structure in topics of the newsgroups dataset.

The dataset is available for download at http://people.csail.mit.edu/jrennie/20Newsgroups/ (homepage for the newsgroups dataset). It is available in 3 versions for our work we have chosen the original unedited version.

The 20 Newsgroups data set is a collection of 20,000 news articles. It was originally collected by Ken Lang. The collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. comp.sys.ibm.pc.hardware/ comp.sys.mac.hardware), while others are highly unrelated (e.g misc.forsale/ soc.religion.christian). Here is a list of the 20 newsgroups, partitioned according to subject matter:

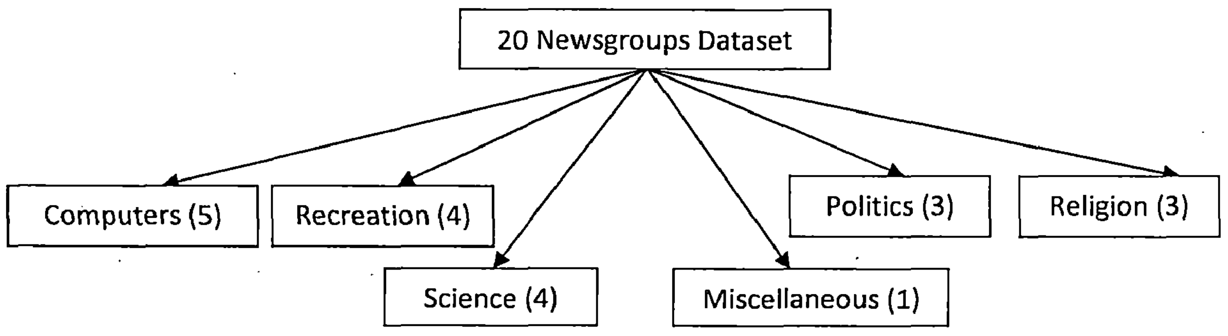


Figure 5.1: 20 Newsgroups contains main categories
(Numbers in braces denote the number of subcategories in each class)

Computers	**Recreation**	**Science**
Graphics MS Windows IBM Hardware MAC Hardware Windows X	Autos Motorcycles Sports-Baseball Sports-Hockey	Cryptography Electronics Medicine Space
Miscellaneous	**Politics**	**Religion**
Forsale	Guns Mideast Miscellaneous	Miscellaneous Atheism Christianity

Table 5.1: 20 Newsgroups sub-categories under 6 main categories

The above table 5.1 shows the presence of hierarchy in our chosen dataset. Figure 5.1 shows the 6 main categories and the table below it shows the subcategories that exist in each of the category.

5.2 Performance Comparison with other methods

In this section we will discuss the performance of our algorithm on the 20 Newsgroup dataset, and compare the accuracy of our method with other methods used on the same dataset. The results are calculated and compared on the basis of two different metrics accuracy and f-score, which we have explained in chapter 2 given in eq. 2.15 and eq. 2.16

Accuracy, F-Score rank the clustering quality from zero (worst) to one (best). The Accuracy value will be the one when every document is distributed to its corresponding dominant cluster. The F-Score value will be one when every category has a corresponding cluster containing the same set of documents.

The results have been taken from the paper "Text Clustering via Particle Swarm Optimization" [18]. The results shown in the following graph have been calculated based on the given dataset shown in table 5.2-

S. NO.	Source Category		No. of Documents	k (number of clusters)
	Category	Sub-Category		
1	Computers Religion	Graphics Atheism	1000 1000	2
2	Politics Politics	Mideast Misc.	1000 1000	2
3	Computers Recreation Science Religion	Graphics Sports-Hockey Cryptography Misc	1000 1000 1000 1000	4
4	Religion Recreation Politics Politics	Atheism Sports-Baseball Guns Misc.	1000 1000 1000 1000	4

Table 5.2: Database division for our experiments

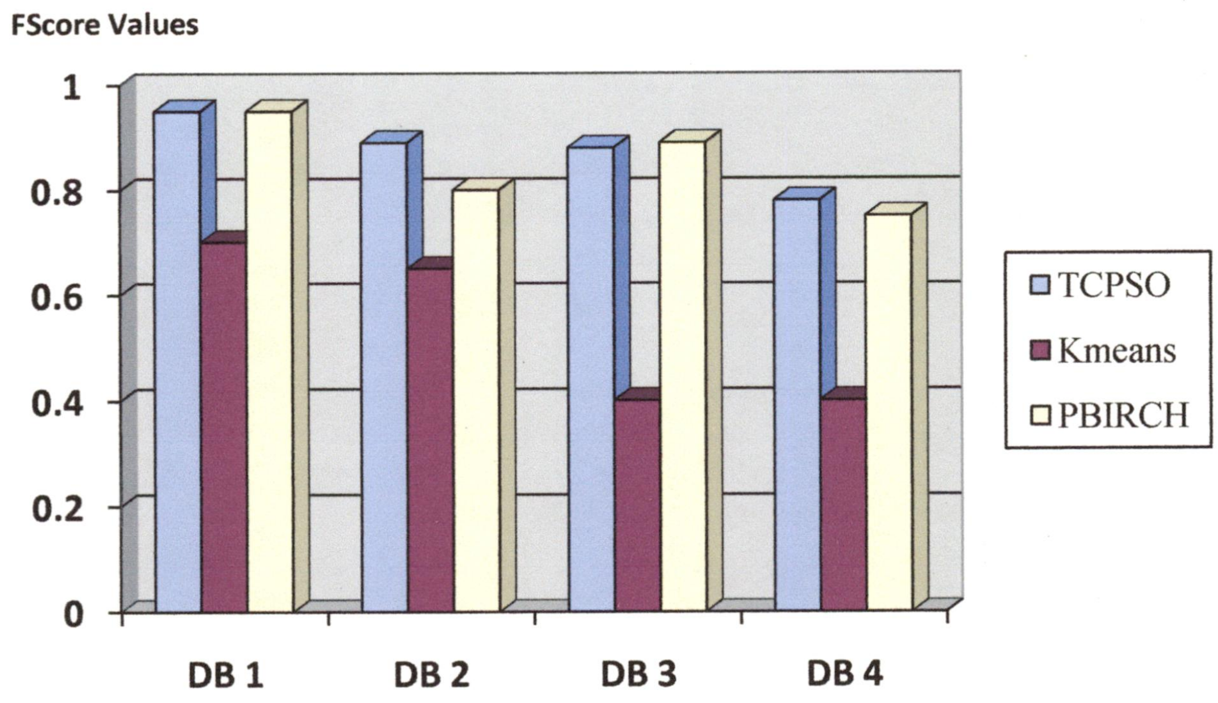


Figure 5.2: Comparison of results of our method with TCPSO and Kmeans

The experiments showed that the method worked accurately for widely separated classes, for classes with fuzzy topics the clusters were not accurate but the accuracy is still comparable to other methods used on the same dataset.

The results show that our method works better than other methods in presence of widely separated categories in the database for example in case of DB1 and DB3 shown in figure 5.2. This is because the clusters formed at each level of hierarchy are also well separated from each other. In the presence of a miscellaneous class or on including multiple sub categories from the same main category (as in DB 2 and DB 4 in above figure 5.2) results in a slightly poorer quality clusters because of the use of distance metric as similarity measure. Distance when used as a similarity measure fails to clearly separate the documents belonging to similar categories because many of the words are common between documents of both sub categories.

We conducted experiments for accuracy of our method on several other combinations of categories as well. The result of which is shown in the table given below-

S. NO.	Source Category	No. of Documents	Level 1		Level 2	
			No. of clusters	% age accuracy	No. of clusters	% age accuracy
1	**2 Categories** Computer (Graphics) Religion (Atheism)	2000	2	95	-	-
2	**4 Categories** Religion (Atheism) Sports (Baseball) Politics (Guns, Misc)	4000	4	91	-	-
3	**4 Categories, 8 sub-categories** Computers (Graphics, Mac Hardware) Sports (Baseball, Hockey) Science (Medicine, Space) Politics(Guns, Mideast)	8000	4	86	8	79
4	**4 Categories, 12 sub-categories** Computers (Graphics, Mac hardware, Windows X) Recreation (Autos, Baseball, Hockey) Science (Electronics, Medicine, Space) Politics (Guns, Mideast, Misc)	12000	4	78	12	72

Table 5.3: Accuracy of results as we increase the number of sub-classes

5.3 Performance comparison with Parallel Implementation

The study of time complexity of the algorithm hugely depends on the hardware of the machine used. All CUDA and sequential experiments were conducted on a PC with 4 GB RAM and an Intel Core 2 Duo E6750 2.66GHz processor running Windows 7 64-bit. An NVIDIA 8800GTX GPU with 768MB of memory was used for all experiments. All of the CUDA algorithms were written using CUDA version 2.3, while the sequential algorithms were written in C++ using Visual Studio 2010. NVIDIA graphs driver version 197.45 was used for the project. Thrust version 1.2 was used for the applicable algorithms.

The results are based on CUDA's Thrust library implementation of the BIRCH clustering algorithm. Point to note is that although GPU's are capable of very high speed computations but the speedups are generally restrained by the amount of memory transfers required, as all the data cannot be fed into the GPUs at the same rates as it can use it.

The graph here shows how parallelization speedups the clustering-

No. of Documents	Vocabulary Size (No. of words)	Timing for Creating Clustering Tree (In minutes)		Relative Speed-up Obtained
		BIRCH	PBIRCH	
1000	7805	14.3	2.6	5.5
2000	12539	29.5	5.5	5.36
4000	16843	63.1	10.25	6.18
8000	18392	139.8	21.7	6.45
12000	21560	212.5	31.95	6.7

Table 5.4: Speed up obtained with varying number of documents in parallel version over the normal one.

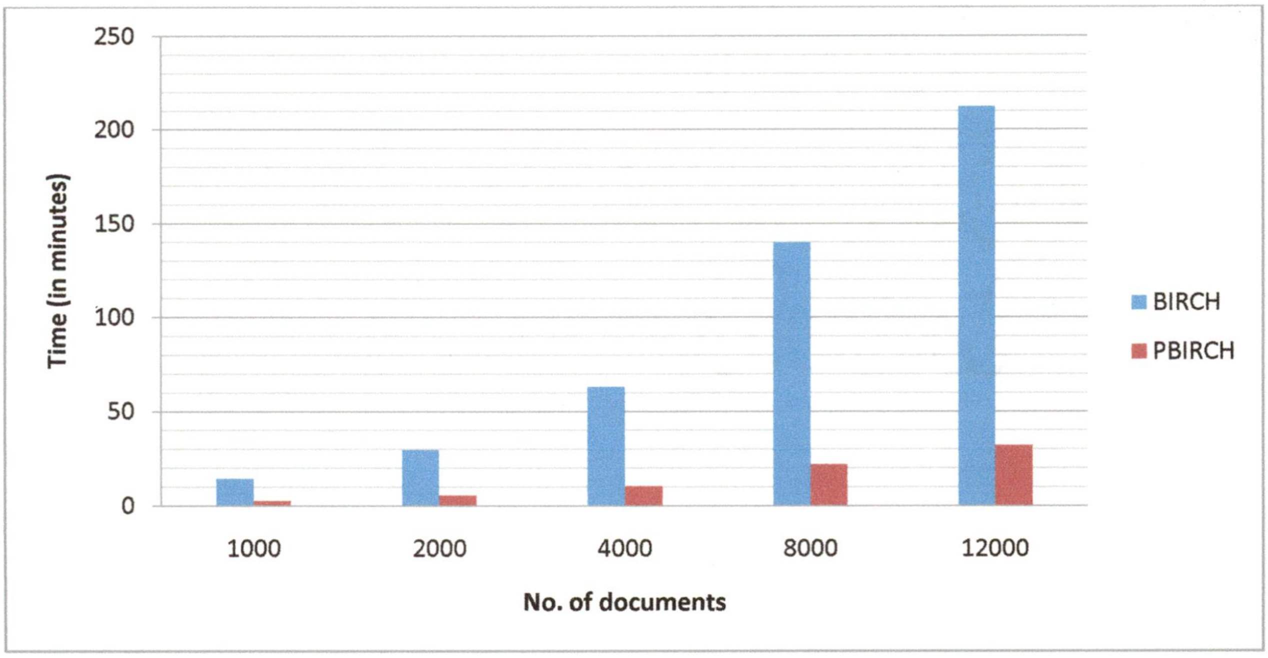


Figure 5.3: Speed up obtained with varying number of documents in parallel BIRCH over the normal implementation.

In figure 5.3 blue bars show the time taken by normal version of the BIRCH to run and red lines show the parallel implementation running time. We can see that the time taken by the parallel version is significantly low and the parallelization shows a straight 5-6 times better performance over normal implementation. Although the speed up is constrained by the amount of time spent in copying the vectors from host to device, but this can be reduced further with newer versions of GPUs available where the device can directly access the host memory and the copying is not required.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this chapter we have two sections in first section we would like to conclude our work by highlighting some of its features and second section we would like to suggest some directions where the future work could be carried on.

6.1 Conclusion

The following conclusions can be drawn from our present work-

1. In our work we have proposed the use of a hierarchical clustering algorithm for unstructured text which uses a hybrid approach of hierarchical clustering and iterative partitioning based clustering methods. The proposed scheme produces hierarchical clusters thus finding similar document groups at various levels of hierarchy.

2. Our scheme uses a tree based compression data structure to store summaries of clusters in place of complete data. The tree thus formed is stored in memory at all points of time so the addition of new item does not require the program to read again any of the previous data values from disk. Thus even with a constraint on main memory we can work with very large sized datasets.

3. In the present work we have parallelized the computations on similarity measurement. These calculations are performed on large sized numeric vectors, with the help of CUDA supportive GPUs these computations are performed much more quickly as compared to serial implementation. Thus the parallelization produces a significant speed up in clustering process.

Our solution is better in terms of both time and space complexity compared to the current methods used to cluster text.
Thus our method can find use in various applications in many basic document organizing tasks like hierarchical organization of news articles, as a search engine add on for more organized results, and as a simple offline document clustering tool.

6.2 Suggestions for Future Work

Our Method works well for data like News articles with differentiating classes, but in the presence of misc articles the accuracy is decreased. For such Fuzzy classes the method could either be combined with some other clustering method at macro clustering in place of simply finding nearest cluster.
Also there is a need to address multi class problem that is a document might belong to multiple classes at the same level of hierarchy, in our method it is simply grouped with the most near match. But in actual there must be some threshold limits to allow documents to belong to multiple clusters.
Also we have proposed to use the normal Distance Metric for Similarity which effects the result when the articles of vastly varying lengths. The Idea of CF tree for compression could be taken a step forward to calculate summaries on the basis of vector representation of document itself. In that case the accuracy of the implemented method would be better even when documents or articles are taken from different sources or are of varying type.

REFERENCES

[1] J. Han and M. Kamber, DATA MINING CONCEPTS AND TECHNIQUES, Morgan Kaufmann, 2006.

[2] A. K. Jain , M. N. Murty , P. J. Flynn, Data clustering: a review, ACM Computing Surveys (CSUR), v.31 n.3, p.264-323, Sept. 1999

[3] Tian Zhang, Raghu Ramakrishnan, Miron Livny; “BIRCH: an efficient data clustering method for very large databases”, In SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data, pp. 103-114, 1996

[4] Na Wang; Pengyuan Wang; Baowei Zhang; , "An improved TF-IDF weights function based on information theory," Computer and Communication Technologies in Agriculture Engineering (CCTAE), 2010 International Conference On , vol.3, no., pp.439-441, 12-13 June 2010.

[5] Lee, D.L.; Huei Chuang; Seamons, K.; , "Document ranking and the vector-space model," Software, IEEE , vol.14, no.2, pp.67-75, Mar/Apr 1997

[6] NVIDIA Corporation, NVIDIA CUDA: Compute unified device architecture, Programming guide 2007, pp. 1–83.

[7] Premalatha K. and Natarajan A.M, A literature review on document clustering, Information Technology Journal 9, 993-1002, 2010

[8] D. Cutting, J. Karger, J. Pedersen, and J. Turkey. Scatter/gather: A cluster-based approach to browsing large document collections. In Proceedings of the Fifteenth International Conference on Research and Development in Information Retrieval, pages

318–329, Copenhagen, Denmark, June 1992.

[9] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In Research and Development in Information Retrieval, pages 46–54, 1998

[10] Zhexue Huang. "A Fast Clustering Algorithm to Cluster very Large Categorical Data Sets in Data Mining", DMKD, 1997

[11] Cui, X.; Potok, T.E.; Palathingal, P.; , "Document clustering using particle swarm optimization," Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE, vol. no. , pp. 185- 191, 8-10 June 2005

[12] Csorba, K.; Vajk, I.; "Term Clustering and Confidence Measurement in document .Clustering," Computational Cybernetics, 2006. ICCC 2006. IEEE International Conference on , vol., no., pp.1-6, 20-22 Aug. 2006

[13] Liping Jing, Michael K. Ng, Joshua Zhexue Huang, "An Entropy Weighting k-Means Algorithm for Subspace Clustering of High-Dimensional Sparse Data," IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 8, pp. 1026-1041, June 2007

[14] H. Sun, Z. Liu, and L. Kong, "A Document Clustering Method Based on Hierarchical Algorithm with Model Clustering", in Proc. AINA Workshops, 2008, pp.1229-1233.

[15] Muflikhah, L.; Baharudin, B.; , "Document Clustering Using Concept Space and Cosine Similarity Measurement," *Computer Technology and Development, 2009. ICCTD '09. International Conference on* , vol.1, no., pp.58-62, 13-15 Nov. 2009

[16] Gil-Garcia R., Pons-Porrata A., Dynamic hierarchical algorithms for document clustering. Pattern Recognition Letters 31 (6), pp.469-477, 2010.

[17] Porter, M.F.: An algorithm for suffix stripping. Program, Vol. 14, No. 3, 1980

[18] Yanping Lu; Shengrui Wang; Shaozi Li; Changle Zhou; , "Text Clustering via Particle Swarm Optimization," *Swarm Intelligence Symposium, 2009. SIS '09. IEEE* , vol., no., pp.45-51, March 30 2009-April 2 2009